Challenge Jan 2023

Christmas Model created by ChatGPT

A solution with DT5GL by Jack Jansonius – 16 February 2023

Problem Statement (from the web site):

This model defines a set of people, a set of gifts, and the happiness level and cost of each gift. The objective is to maximize the total happiness, subject to the budget constraint that the total cost of the gifts must be less than or equal to the budget, and the constraint that each person can only receive one gift.

Here is a sample of test data:
PEOPLE: "Alice", "Bob", "Carol", "Dave", "Eve"
GIFTS: "Book", "Toy", "Chocolate", "Wine", "Flowers"
GIFT COSTS: 10, 20, 5, 15, 7
HAPPINESS:
"Book": [3, 2, 5, 1, 4]
"Toy": [5, 2, 4, 3, 1]
"Chocolate": [1, 3, 4, 5, 2]
"Wine": [2, 5, 3, 4, 1]
"Flowers": [4, 3, 1, 2, 5]
BUDGET: 50

1. Problem analysis

Putting the test data into a table makes it clear that the problems in this allocation issue only begin fully with a lower chosen budget, say 35.

	5 ➔	4 ➔	3 ➔	2 ➔	1
Alice	Toy:20	Flowers: 7	Book: 10	Wine:15	Chocolate: 5
Bob	Wine:15		Chocolate: 5 Flowers: 7	Book: 10 Toy:20	
Carol	Book: 10	Toy:20 Chocolate: 5	Wine:15		Flowers: 7
Dave	Chocolate: 5	Wine:15	Toy:20	Flowers: 7	Book: 10
Eve	Flowers: 7	Book: 10		Chocolate: 5	Toy:20 Wine:15

What is remarkable here is that a person doesn't need to have a gift preference for each happiness level and can also have multiple gift preferences for one happiness level. There is no need to be cumbersome about the latter: in that case, the most expensive gift can simply be deleted.

A very simple algorithm[1] can now be used to reason out the optimal solution for any budget:

```
A: Give all persons what they want most.
B: If the cost of this is within the budget, then done.
   Otherwise: replace the gift with a person by a gift with
              the smallest decrease in happiness and the largest decrease in cost.
              Repeat B.
```

If we give all persons what they want most, the sum of the gifts is 57, which is not within the budget of 50.

Now it makes no sense to replace a gift with happiness level 5 with a gift with a lower happiness level which is also more expensive. These gifts can actually be crossed out!

	5 ➔	4 ➔	3 ➔	2 ➔	1
Alice	Toy:20	Flowers: 7	Book: 10	Wine:15	Chocolate: 5
Bob	Wine:15		Chocolate: 5 ~~Flowers: 7~~	Book: 10 ~~Toy:20~~	
Carol	Book: 10	~~Toy:20~~ Chocolate: 5	Wine:15		Flowers: 7
Dave	Chocolate: 5	Wine:15	Toy:20	Flowers: 7	Book: 10
Eve	Flowers: 7	Book: 10		Chocolate: 5	~~Toy:20~~ Wine:15

So the first selection is as follows:

Alice: Toy → Flowers; happiness reduction: 1; cost reduction: 13
Bob: Wine → Chocolate: happiness reduction: 2; cost reduction: 10
Carol: Book → Chocolate: happiness decrease: 1; cost reduction: 5
Dave: No substitution possible!
Eve: Flowers → Chocolate: happiness reduction: 3; cost reduction: 2

Clearly, Alice is the 'winner' here; she gets Flowers instead of a Toy, which brings the total happiness level to 24 and the total cost to 44. Which is therefore the optimal solution for a budget of 50.

If we set the budget at 40; then the next person will have to turn in their most desirable gift. Note that Alice just rejoins the selection:

Alice: Flowers → Chocolate: happiness reduction: 3; cost reduction: 2
Bob: Wine → Chocolate: happiness reduction: 2; cost reduction: 10
Carol: Book → Chocolate: happiness decrease: 1; cost reduction: 5
Dave: No substitution possible!
Eve: Flowers → Chocolate: happiness reduction: 3; cost reduction: 2

[1] It is remarkable that a number of submissions to this challenge did not contain algorithm(s), but a mathematical model (so that there is in fact no "decision modeling" here). The solutions that do use algorithms are actually incomprehensible to an outsider. My solution is based on a combination of decision tables (5GL) and SQL (4GL), as will be shown later. I am curious if an algorithmic solution to this challenge is also possible based on a 3GL (Python, for example) or a 4GL without more. And whether ChatGPT will ever come up with such an algorithmic solution.

Now Carol can turn in her book for chocolate, which brings the total cost to 39 so an optimal solution is found when the budget is 40.

If we set the budget below 39 again, then it is Bob's turn which again costs 2 happiness points with a cost reduction of 10.

At this point (with a budget lower than 29) there are still 2 people in the race, namely Alice and Eve; the others have already reached their preliminary end point, namely: chocolate.

And that choice looks as follows:
Alice: Flowers → Chocolate: happiness reduction: 3; cost reduction: 2
Eve: Flowers → Chocolate: happiness reduction: 3; cost reduction: 2

Now the choice is completely arbitrary[2] and Eve is chosen based on sorting.

If the budget falls below 27, all persons receive chocolate and the happiness level drops to 15 points.

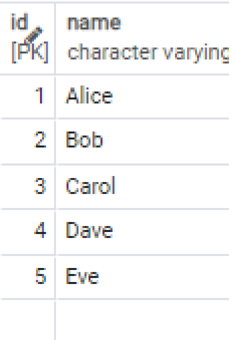
If the budget falls below 25, then the first persons can also return their chocolate, in order of increasing happiness loss.

[2] This is exactly the solution, as found by Hakan Kjellerstrand for a budget of 27; see: http://hakank.org/picat/christmas_model.pi. However, I make no attempt to understand this solution. I also do not have something built in to detect these 2 optimal solutions.

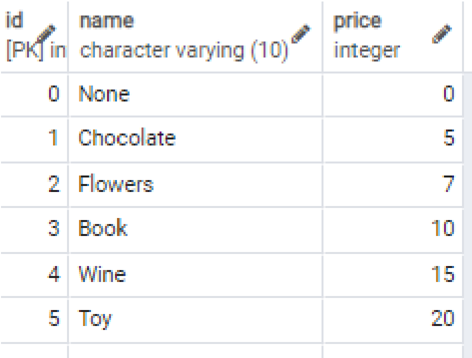
2. Implementation

In order to realize the foregoing, a preparation takes place first. For this purpose, a table is used, where the first 3 fields are derived from the test data of the challenge and the next 3 fields contain the information for determining the optimal solution within a given budget.

In addition to the tables **person** and **gift**:

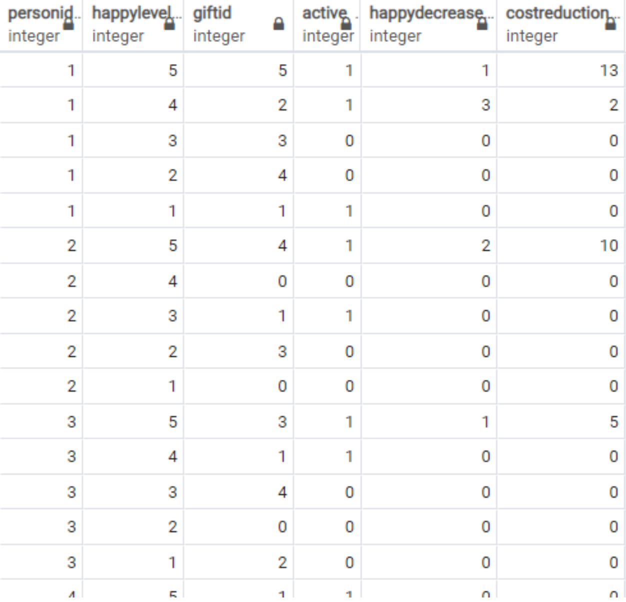


id [PK]	name character varyinç
1	Alice
2	Bob
3	Carol
4	Dave
5	Eve



id [PK] in	name character varying (10)	price integer
0	None	0
1	Chocolate	5
2	Flowers	7
3	Book	10
4	Wine	15
5	Toy	20

there is a table **preference**, which after running preprocessing contains the following data:



personid.. integer	happylevel.. integer	giftid integer	active integer	happydecrease.. integer	costreduction.. integer
1	5	5	1	1	13
1	4	2	1	3	2
1	3	3	0	0	0
1	2	4	0	0	0
1	1	1	1	0	0
2	5	4	1	2	10
2	4	0	0	0	0
2	3	1	1	0	0
2	2	3	0	0	0
2	1	0	0	0	0
3	5	3	1	1	5
3	4	1	1	0	0
3	3	4	0	0	0
3	2	0	0	0	0
3	1	2	0	0	0
4	5	1	1	0	0

Determining which person's turn for a gift with the least happydecrease and the highest costreduction is now nothing more than sorting this table by happydecrease (from low to high), and within that by costreduction (from high to low). But only for persons with 2 or more gifts active. Persons who have only 1 gift active have already reached their endpoint (namely chocolate) and thus are not included in the sorting; this is immediately the case for Dave.

Implementation of the decision model in DT5GL (Preperation):

```
PostgreSQL_database: "christmas"

Table 0:
If:                                              | 0| 1|
'Next person present'                            | Y| N|
Then:
Nextperson is Selected                           | X|  |
Nextperson is Finished                           |  | X|
# .......
# Repeat until: finished

Proposition: 'Next person present'
Obtain_instance_from_database_view: person


Table 1:
If:                                              | 0| 1| 2| 3| 4|
'Next preference for person'                     | Y| Y| Y| Y| N|
current.giftid > 0                               | Y| Y| Y| N| -|[3]
previous.personid > 0                            | Y| Y| N| -| -|
previous.price > current.price                   | Y| N| -| -| -|
Then:
Action is UpdatePreviousAndCurrent               | X|  |  |  |  |
Action is MakeCurrentInactive                    |  | X|  | X|  |
Action is InitPrevious                           |  |  | X|  |  |
Action is Finished                               |  |  |  |  | X|
# .......
# Repeat until: Finished

Proposition: 'Next preference for person'
Obtain_instance_from_database_view: current

Attribute: happydecrease  Type: Integer
Equals: previous.happylevel - current.happylevel

Attribute: costreduction  Type: Integer
Equals: previous.price - current.price


Attribute: current.happylevel     Type: Integer
Attribute: current.price          Type: Integer
Attribute: previous.happylevel    Type: Integer
Attribute: previous.price         Type: Integer
```

[3] This condition still looks somewhat technical and may soon be replaced by a proposition, for example: 'This preference contains a gift.' Further down should then be specified:
Proposition: 'This preference contains a gift'
Equals: current.giftid > 0

In this way, all conditions in this table can be replaced by propositions.

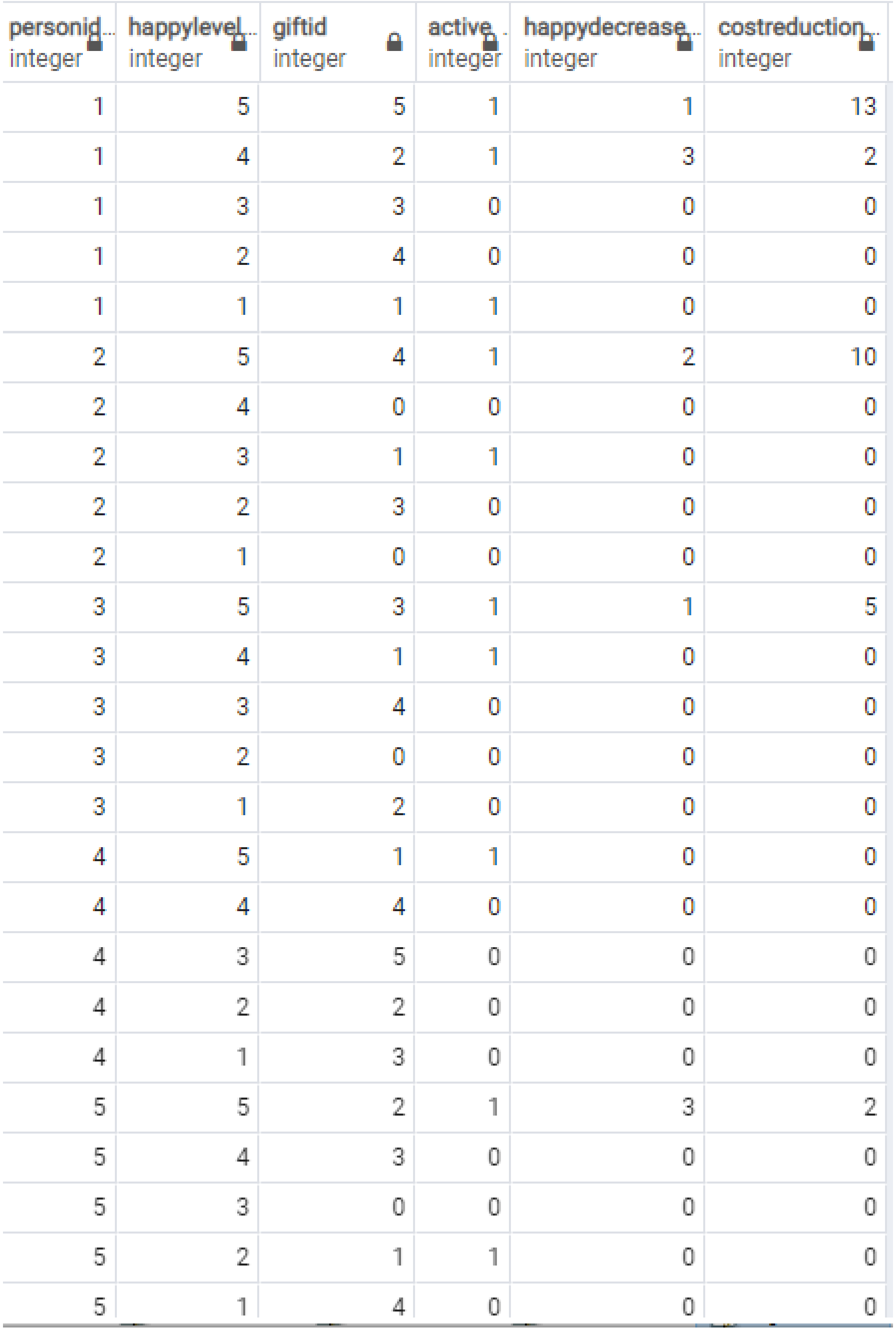
```
########################### Database views ###########################

Database_view: person
With_attributes: id, name
Query:
 SELECT id, name
  FROM person
 LIMIT 1 OFFSET %s
With_arguments: person.auto_index


Database_view: current
With_attributes: personid, happylevel, giftid, price
Query:
    SELECT a.personid AS personid,
           a.happylevel AS happylevel,
           a.giftid AS giftid,
           b.price AS price
    FROM preference AS a
        JOIN gift AS b ON (a.giftid = b.id)
    WHERE a.personid = %s
    ORDER BY happylevel DESC
    LIMIT 1 OFFSET %s
With_arguments: person.id, current.auto_index


Initial_database_table: init_previous
Query:
    CREATE TEMP TABLE previous AS
    SELECT 0 AS personid,
           0 AS happylevel,
           0 AS price
End_Query

Database_view: previous
With_attributes:
personid, happylevel, price
Query:
SELECT *
  FROM previous
 LIMIT 1
End_Query
```

```
########################### GoalAttributes ###########################

GoalAttribute: Nextperson
Repeat_until: Finished

Case: Finished
Print: "===================================================================="
Print: "Preference update finished."
Print: "===================================================================="

Case: Selected
Print: "#REM# -- print nothing"
>SQL:  "UPDATE previous "
-SQL:  "   SET personid = 0,  "
-SQL:  "       happylevel = 0,  "
<SQL:  "       price = 0   "


GoalAttribute: Action
Repeat_until:  Finished

Case: Finished
Print: "preferences for %s updated."   person.name

Case: MakeCurrentInactive
Print: "#REM# -- print nothing"

Case: UpdatePreviousAndCurrent
Print: "#REM# -- print nothing"

>SQL:  "UPDATE preference "
-SQL:  "   SET happydecrease = %s,  "  happydecrease
-SQL:  "       costreduction = %s   "  costreduction
-SQL:  " WHERE personid = %s  "        previous.personid
<SQL:  "   AND happylevel = %s  "      previous.happylevel

>SQL:  "UPDATE previous "
-SQL:  "   SET happylevel = %s,  "     current.happylevel
<SQL:  "       price = %s   "          current.price

>SQL:  "UPDATE preference "
-SQL:  "   SET active = 1  "
-SQL:  " WHERE personid = %s  "        current.personid
<SQL:  "   AND happylevel = %s  "      current.happylevel


Case: InitPrevious
Print: "#REM# -- print nothing"

>SQL:  "UPDATE previous "
-SQL:  "   SET personid = %s,  "       current.personid
-SQL:  "       happylevel = %s,  "     current.happylevel
<SQL:  "       price = %s   "          current.price

>SQL:  "UPDATE preference "
-SQL:  "   SET active = 1  "
-SQL:  " WHERE personid = %s  "        current.personid
<SQL:  "   AND happylevel = %s  "      current.happylevel
```

Testrun Preperation

```
preferences for Alice updated.
preferences for Bob updated.
preferences for Carol updated.
preferences for Dave updated.
preferences for Eve updated.
====================================================================
Preference update finished.
====================================================================
Time elapsed: 0:00:03.430148
```

Table Preference now contains:



personid...	happylevel...	giftid	active...	happydecrease...	costreduction...
integer	integer	integer	integer	integer	integer
1	5	5	1	1	13
1	4	2	1	3	2
1	3	3	0	0	0
1	2	4	0	0	0
1	1	1	1	0	0
2	5	4	1	2	10
2	4	0	0	0	0
2	3	1	1	0	0
2	2	3	0	0	0
2	1	0	0	0	0
3	5	3	1	1	5
3	4	1	1	0	0
3	3	4	0	0	0
3	2	0	0	0	0
3	1	2	0	0	0
4	5	1	1	0	0
4	4	4	0	0	0
4	3	5	0	0	0
4	2	2	0	0	0
4	1	3	0	0	0
5	5	2	1	3	2
5	4	3	0	0	0
5	3	0	0	0	0
5	2	1	1	0	0
5	1	4	0	0	0

Implementation of the decision model in DT5GL (Main):

```
PostgreSQL_database: "christmas"


rTable 0:
If:                                              | 0| 1| 2|
actualgifts.total_costs <= budget                | Y| N| N|
'person_with_2_or_more_active_gifts'             | -| Y| N|
'person_with_1_active_gift'                      | -| -| Y|
Then:
Action is SolutionFound                          | X|  |  |
Action is RemoveHigherPricedGift                 |  | X|  |
Action is RemoveFinalGift                        |  |  | X|
# .......
# Repeat until: SolutionFound


Attribute: actualgifts.total_costs     Type: Integer
Attribute: actualgifts.total_hapiness  Type: Integer
Attribute: restrict.happydecrease      Type: Integer
Attribute: restrict.costreduction      Type: Integer


Attribute: budget  Type: Integer
Equals: 50

Attribute: new_happylevel            Type: Integer
Equals: actualgifts.total_hapiness - restrict.happydecrease

Attribute: new_totalcosts            Type: Integer
Equals: actualgifts.total_costs - restrict.costreduction


Proposition: 'person_with_2_or_more_active_gifts'
Obtain_instance_from_database_view: restrict

Proposition: 'person_with_1_active_gift'
Obtain_instance_from_database_view: final


########################### Database views ###########################

Database_view: actualgifts
With_attributes: total_hapiness, total_costs
Query:
    SELECT coalesce(sum(a.happylevel), 0) as total_hapiness,
           coalesce(sum(b.price), 0) as total_costs
    FROM preference AS a
        JOIN gift AS b ON a.giftid = b.id
    WHERE a.happylevel =(SELECT max(happylevel)
                            FROM preference
                            WHERE personid = a.personid AND
                                active = 1)
End_Query
```

```
Database_view: restrict
With_attributes: personid, personname, happylevel, price, giftname, happydecrease,
costreduction
Query:
    SELECT personid,
        person.name AS personname,
        happylevel AS happylevel,
        price,
        gift.name AS giftname,
        happydecrease,
        costreduction
    FROM preference AS a
        JOIN gift ON a.giftid = gift.id
        JOIN person ON a.personid = person.id
    WHERE active = 1 AND
        /* only persons with 2 or more active gifts */
        personid IN (SELECT personid
                        FROM preference
                        WHERE active = 1
                        GROUP BY personid
                        HAVING count(*) > 1) AND
        happylevel = (SELECT MAX(happylevel)
                        FROM preference AS b
                        WHERE b.personid = a.personid AND
                                active = 1)
    ORDER BY happylevel DESC,
            happydecrease ASC,
            costreduction DESC,
            personid ASC
    LIMIT 1
End_Query

# restrict = persons with multiple active gifts in order of decreasing happiness
level and decreasing cost. The query returns the gift with the highest happiness
level that is replaceable by a gift with lower happiness level and lower cost.


Database_view: final
With_attributes: personid, personname, happylevel, price, giftname, happydecrease,
costreduction
Query:
    SELECT personid,
        person.name,
        happylevel,
        price,
        gift.name,
        happydecrease,
        costreduction
    FROM preference
        JOIN gift ON preference.giftid = gift.id
        JOIN person ON preference.personid = person.id
    WHERE active = 1 AND
        /* only persons with exactly 1 active gift */
        personid IN (SELECT personid
                        FROM preference
                        WHERE active = 1
                        GROUP BY personid
                        HAVING count(*) = 1)
    ORDER BY happylevel ASC
    LIMIT 1
End_Query

# final = persons who can still get only 1 gift, namely the cheapest one.
```

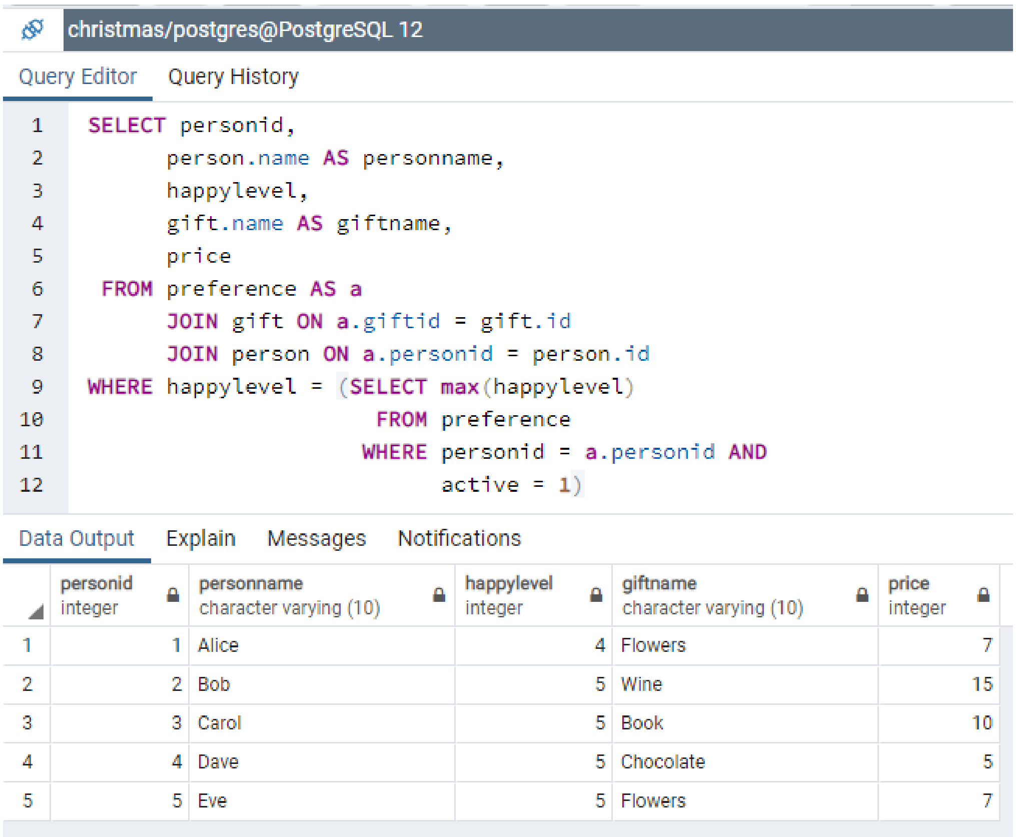
```
########################### GoalAttributes ###########################

GoalAttribute: Action
Repeat_until: SolutionFound

Case: SolutionFound
Print: "===================================================================="
Print: "Opimal solution found."
Print: "Total happiness level: %s" actualgifts.total_hapiness
Print: "Total costs: %s (within the budget: %s)" actualgifts.total_costs budget
Print: "===================================================================="

Case: RemoveHigherPricedGift
Print: "%s (%s) : %s => cheaper gift (-%s) with lower happiness level (-%s)."
restrict.personname restrict.personid restrict.giftname restrict.costreduction
restrict.happydecrease
Print: "        .... new total costs: %s        and hapiness level: %s "
new_totalcosts  new_happylevel
>SQL:  "UPDATE preference "
-SQL:  "   SET active = 0  "
-SQL:  " WHERE personid = %s  "       restrict.personid
<SQL:  "   AND happylevel = %s  "     restrict.happylevel

Case: RemoveFinalGift
Print: "%s gets nothing; cost reduction: %s and lowering happiness level with %s "
final.personname  final.price  final.happylevel
>SQL:  "UPDATE preference "
-SQL:  "   SET active = 0  "
-SQL:  " WHERE personid = %s  "       final.personid
<SQL:  "   AND happylevel = %s  "     final.happylevel
```

Testruns Main

Budget = 50:

```
Alice (1) : Toy => cheaper gift (-13) with lower happiness level (-1).
        .... new total costs: 44        and hapiness level: 24
====================================================================
Opimal solution found.
Total happiness level: 24
Total costs: 44 (within the budget: 50)
====================================================================
Time elapsed: 0:00:02.276971
```

Who gets what:



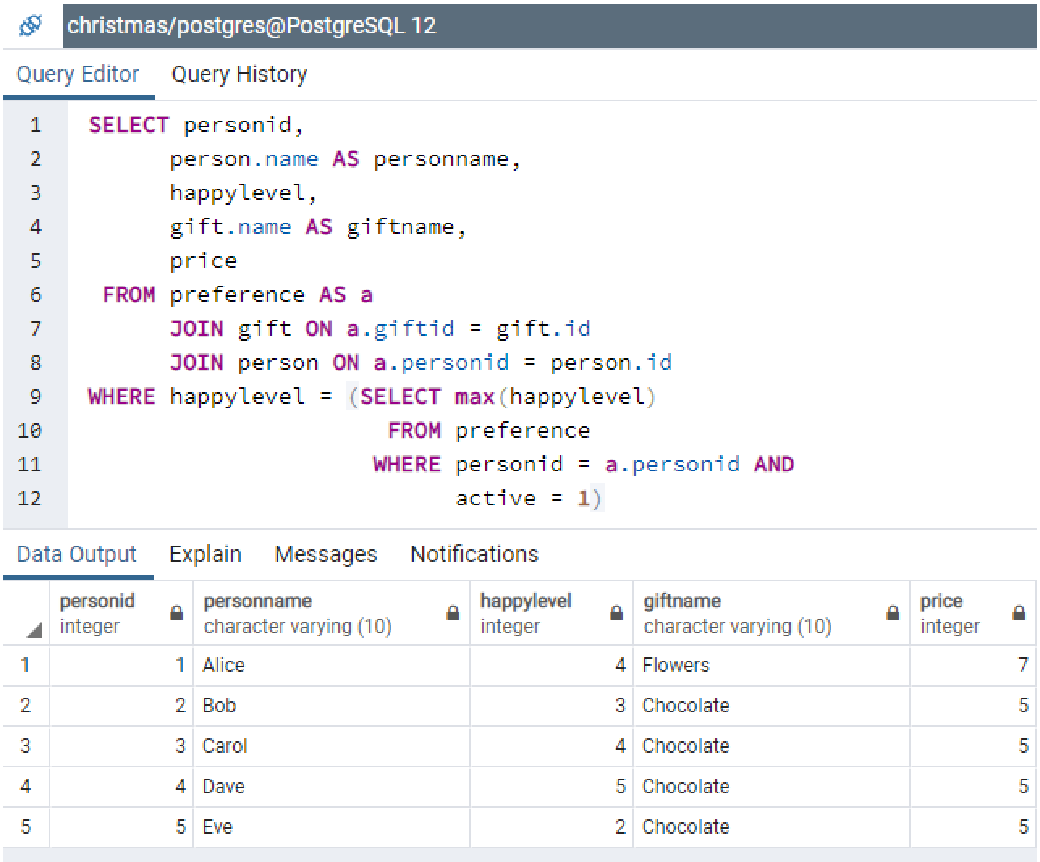
```sql
SELECT personid,
       person.name AS personname,
       happylevel,
       gift.name AS giftname,
       price
 FROM preference AS a
       JOIN gift ON a.giftid = gift.id
       JOIN person ON a.personid = person.id
WHERE happylevel = (SELECT max(happylevel)
                      FROM preference
                     WHERE personid = a.personid AND
                           active = 1)
```

	personid integer	personname character varying (10)	happylevel integer	giftname character varying (10)	price integer
1	1	Alice	4	Flowers	7
2	2	Bob	5	Wine	15
3	3	Carol	5	Book	10
4	4	Dave	5	Chocolate	5
5	5	Eve	5	Flowers	7

Budget = 28:

```
Alice (1) : Toy => cheaper gift (-13) with lower happiness level (-1).
        .... new total costs: 44        and hapiness level: 24
Carol (3) : Book => cheaper gift (-5) with lower happiness level (-1).
        .... new total costs: 39        and hapiness level: 23
Bob (2) : Wine => cheaper gift (-10) with lower happiness level (-2).
        .... new total costs: 29        and hapiness level: 21
Eve (5) : Flowers => cheaper gift (-2) with lower happiness level (-3).
        .... new total costs: 27        and hapiness level: 18
====================================================================
Opimal solution found.
Total happiness level: 18
Total costs: 27 (within the budget: 28)
====================================================================
Time elapsed: 0:00:02.485577
```

Who gets what:



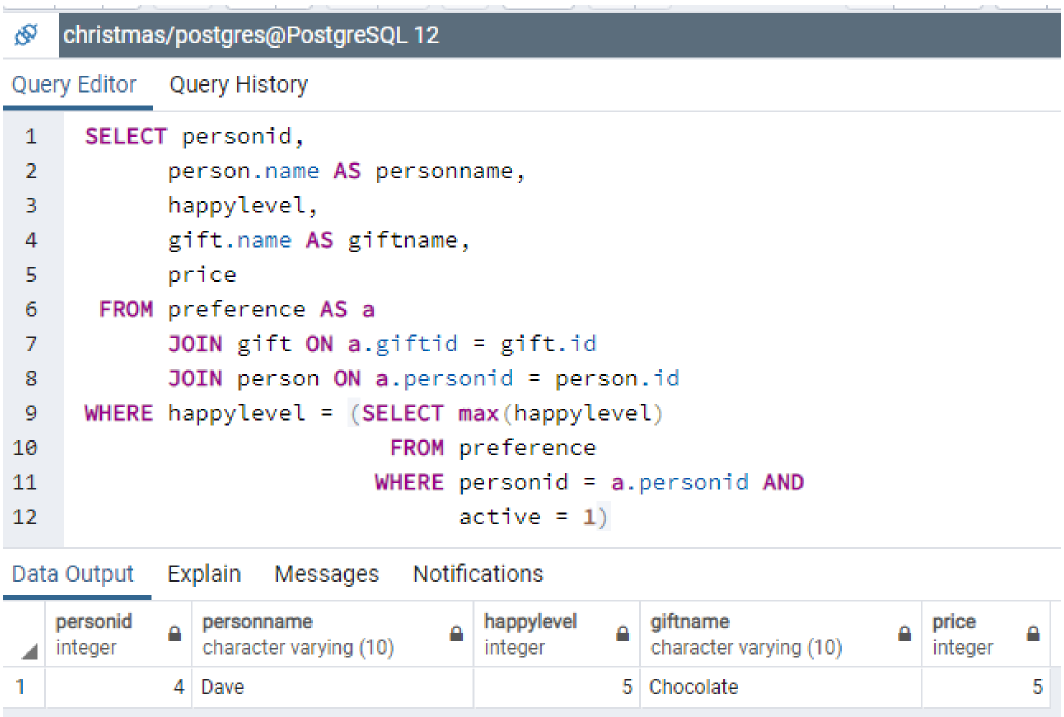
```sql
SELECT personid,
      person.name AS personname,
      happylevel,
      gift.name AS giftname,
      price
 FROM preference AS a
      JOIN gift ON a.giftid = gift.id
      JOIN person ON a.personid = person.id
WHERE happylevel = (SELECT max(happylevel)
                      FROM preference
                     WHERE personid = a.personid AND
                           active = 1)
```

	personid integer	personname character varying (10)	happylevel integer	giftname character varying (10)	price integer
1	1	Alice	4	Flowers	7
2	2	Bob	3	Chocolate	5
3	3	Carol	4	Chocolate	5
4	4	Dave	5	Chocolate	5
5	5	Eve	2	Chocolate	5

Budget = 7:

```
Alice (1) : Toy => cheaper gift (-13) with lower happiness level (-1).
        .... new total costs: 44        and hapiness level: 24
Carol (3) : Book => cheaper gift (-5) with lower happiness level (-1).
        .... new total costs: 39        and hapiness level: 23
Bob (2) : Wine => cheaper gift (-10) with lower happiness level (-2).
        .... new total costs: 29        and hapiness level: 21
Eve (5) : Flowers => cheaper gift (-2) with lower happiness level (-3).
        .... new total costs: 27        and hapiness level: 18
Alice (1) : Flowers => cheaper gift (-2) with lower happiness level (-3).
        .... new total costs: 25        and hapiness level: 15
Alice gets nothing; cost reduction: 5 and lowering happiness level with 1
Eve gets nothing; cost reduction: 5 and lowering happiness level with 2
Bob gets nothing; cost reduction: 5 and lowering happiness level with 3
Carol gets nothing; cost reduction: 5 and lowering happiness level with 4
====================================================================
Opimal solution found.
Total happiness level: 5
Total costs: 5 (within the budget: 7)
====================================================================
Time elapsed: 0:00:02.314122
```

Who gets what:



```sql
SELECT personid,
      person.name AS personname,
      happylevel,
      gift.name AS giftname,
      price
 FROM preference AS a
      JOIN gift ON a.giftid = gift.id
      JOIN person ON a.personid = person.id
WHERE happylevel = (SELECT max(happylevel)
                      FROM preference
                     WHERE personid = a.personid AND
                           active = 1)
```

	personid integer	personname character varying (10)	happylevel integer	giftname character varying (10)	price integer
1	4	Dave	5	Chocolate	5

A brief comment on an earlier design of the main program.

An earlier design of the main program - where I had manually filled all the fields of the "preference" table first - was rather strongly inspired by a solution I had submitted for the "Rebooking Passengers from Cancelled Flights" - challenge of October 2016[4].
So in my first design for this challenge, I again used a kind of bubble sort:

```
rTable 0:
If:                                                   | 0| 1| 2|
actualgifts.total_costs <= budget                     | Y| N| N|
'person_with_2_or_more_active_gifts'                  | -| Y| N|
'person_with_1_active_gift'                           | -| -| Y|
Then:
PreAction is SolutionFound                            | X|  |  |
PreAction is OptimalGiftSwap                          |  | X|  |
PreAction is RemoveFinalGift                          |  |  | X|
# .......
# Repeat until: SolutionFound

Table 1:
If:                                                   | 0| 1| 2| 3|
PreAction.getvalue is OptimalGiftSwap                 | Y| Y| Y| N|
'next_person_with_2_or_more_active_gifts'             | Y| Y| N| -|
'Next giftchange has higher prio'                     | Y| N| -| -|
Then:
Action is Swap                                        | X|  |  |  |
Action is NoSwap                                      |  | X|  |  |
Action is ChangeGift                                  |  |  | X|  |
Action is Finished                                    |  |  |  | X|
# .......
# Repeat until: Finished, ChangeGift


rTable 2:
If:                                                   | 0| 1| 2|
restrict.happydecrease < current.happydecrease        | Y| N| N|
restrict.happydecrease = current.happydecrease        | -| Y| Y|
restrict.costreduction > current.costreduction        | -| Y| N|
restrict.costreduction = current.costreduction        | -| -| Y|
restrict.happylevel    > current.happylevel           | -| -| Y|
Then:
'Next giftchange has higher prio'                     | X| X| X|
# .......
```

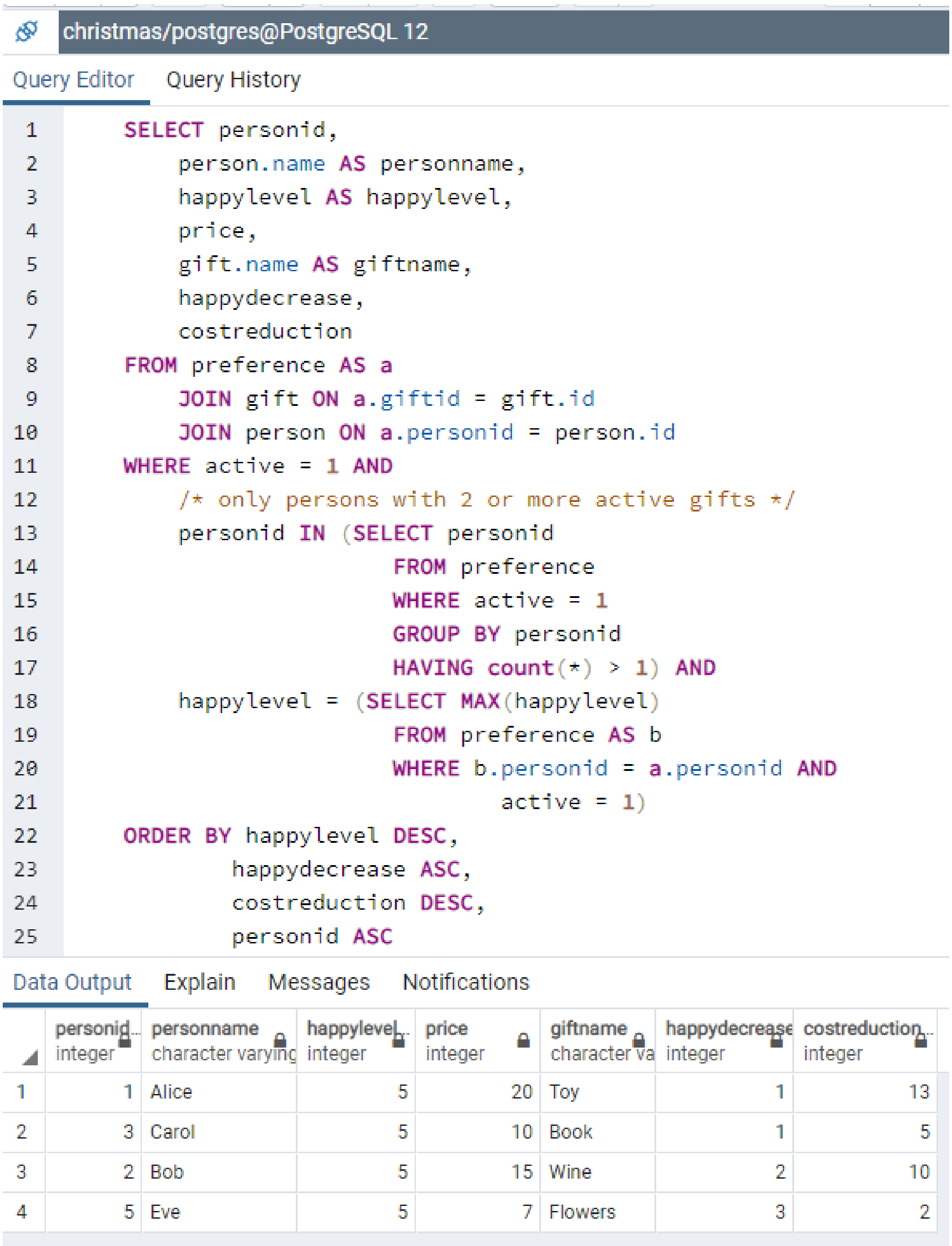
Very nice of course that this works, but it is much more efficient to let SQL itself do this sorting! So with that, I could delete tables 1 and 2 again (and the same applies to the solution I had submitted for October 2016).

The desired sorting is included in the query for the database view 'restrict':

```
    ORDER BY happylevel DESC,
             happydecrease ASC,
             costreduction DESC,
             personid ASC
```

[4] https://dmcommunity.org/challenge/challenge-oct-2016/

After running the preparation, below is a list of persons eligible for a cheaper gift with lower happiness level, then with LIMIT 1 the first one is passed to the program:



```
SELECT personid,
    person.name AS personname,
    happylevel AS happylevel,
    price,
    gift.name AS giftname,
    happydecrease,
    costreduction
FROM preference AS a
    JOIN gift ON a.giftid = gift.id
    JOIN person ON a.personid = person.id
WHERE active = 1 AND
    /* only persons with 2 or more active gifts */
    personid IN (SELECT personid
                    FROM preference
                    WHERE active = 1
                    GROUP BY personid
                    HAVING count(*) > 1) AND
    happylevel = (SELECT MAX(happylevel)
                    FROM preference AS b
                    WHERE b.personid = a.personid AND
                            active = 1)
ORDER BY happylevel DESC,
        happydecrease ASC,
        costreduction DESC,
        personid ASC
```

	personid... integer	personname character varying	happylevel... integer	price integer	giftname character va	happydecrease integer	costreduction... integer
1	1	Alice	5	20	Toy	1	13
2	3	Carol	5	10	Book	1	5
3	2	Bob	5	15	Wine	2	10
4	5	Eve	5	7	Flowers	3	2